DIGITAL MEASUREMENTS OF SPREAD SPECTRUM CLOCKING

Background Of The Invention

The FCC in the United States and other regulatory bodies in foreign jurisdictions have become more aggressive in setting performance standards for the radiation of EMI (Electro-Magnetic Interference). Digital equipment that operates according to clock signals, such as computers and their mass storage peripherals, have been identified as a source of EMI. Historically, a virtue of a clock signal was frequency stability; many data transmission systems have crystal controlled clocks. The combination of this frequency stability and the fast edges of the clock and its associated signals combine to produce pronounced collections of harmonics that can be the cause of EMI. For various reasons, it might not be technically or economically feasible to prevent the EMI through shielding. One of the techniques that has been developed to reduce EMI from such systems is to apply spread spectrum techniques to the clock system. SSC (Spread Spectrum Clocking) is a technique where the frequency of the clock signal is made to vary with time in some controlled manner.

For example, there is a Serial ATA standard for hard disc drives that specifies that the clock frequency of 75 MHZ (or 66 MHZ) vary with respect to time downwards and then back up again by one half percent, and at a rate that varies from 30 KHz to 33 KHz. That is, the clock frequency undergoes a varying degree of Frequency Modulation (FM). The idea is to distribute the harmonic content throughout a portion of the spectrum to reduce its power level at any particular frequency.

The FCC and the other regulatory bodies have certification requirements that require the manufacturer to measure the performance of the system to ensure that it meets the regulatory requirements. Items of interest in a system using SSC include the degree of power reduction at certain frequencies (e.g., the nominal clock frequency without FM) and a description of the modulating FM signal itself. Now it is not the case that there is no test equipment available to make such measurements; there is. But the circuitry that needs to be tested often resides on an IC (Integrated Circuit) die that will be tested on a system that is designed to probe, power up, stimulate and test die locations on undiced wafers. They are well suited to the task of ordinary functional testing of digital operation, but do not incorporate the kinds of capabilities found in the type of laboratory grade bench test equipment (flexible counters, fast digital oscilloscopes and full featured spectrum analyzers) that would ordinarily be used to make measurements related to SSC. What is more, it would be electrically troublesome, not to mention expensive, to connect such bench test equipment to a wafer or die. Yet it is most desirable in high volume production situations to test parts as early as possible, so as to avoid the expense of building assemblies with bad parts, since the cost of finding and replacing a bad part goes up rapidly as assembly fabrication proceeds. The parts we are interested in do get otherwise tested by a digital IC tester such as the Agilent 93000 SOC System (SOC stands for System On a Chip). However, depending upon how any particular such system is configured, the tester might not be equipped with the hardware needed to perform measurements of RF spectrum and FM characteristics. The Agilent 93000 SOC System can indeed be equipped with such measurement hardware, but to do so involves a certain expense. Such an expense might be viewed as a lamentable inefficiency if it were incurred only for the measurement of parameters related to SSC. It would desirable if there were a way that the measurement capabilities of the Agilent 93000 SOC System, and of other similar systems, could be easily and at a low cost, extended to measure the spectrum, deviation and rates of deviation for an SSC operated system. What to do?

Summary Of The Invention

A solution to the problem of measuring the RF spectrum, and FM characteristics of the clock signal in an SSC operated system is to take advantage of one of the one or more high speed single-bit test channels that can be present in the Agilent 93000 SOC System. Such a channel can be configured to measure, at up to 2.5 Gbps, the logical value of an input signal applied thereto. The SSC signal of interest is applied to one of the high-speed single-bit test channels. A conventional FFT (Fast Fourier Transform) is performed on the captured data to discover the aggregate nature of the distributed spectral components, which can then be compared with associated specifications. The captured data is applied to another algorithm to find the FM modulation profile. That algorithm involves operating on captured digital data to perform a frequency translation operation (equivalent to heterodyning), a first filtering operation, a discrete differentiation operation and raw phase extraction that includes a summation of two components and a root square operation, finally followed by a second filtering operation to produce an FM modulation profile. The algorithm for finding the SSC modulation profile does not require the high-speed digital channel to meet Nyquist sampling requirements.

Brief Description Of The Drawings

Figure 1 is a simplified flow chart of how SSC measurements are made in accordance with an aspect of the invention;

Figures 2A and 2B are spectrum diagrams useful in appreciating the operation of the simplified flow chart of Figure 1; and

Figures 3A and 3B are respective phase and frequency diagrams illustrating the recovery of an FM modulation profile from an SSC signal that has been processed to extract phase changes as a function of time.

Description Of A Preferred Embodiment

Refer now to Figure 1, wherein is shown a simplified flow chart 1 that includes an algorithm for obtaining the FM parameters of an SSC signal from captured digital data that does not necessarily meet Nyquist criteria. The first step 2 in the flow chart 1 is the capture of the SSC data using the "digital channel" capability of the Agilent 93000 SOC System. In this type of measurement an SSC signal is applied to a comparator having a threshold and the logical results of TRUE and FALSE appear as a stream of digital data at that matches the SSC signal. A digital "1" will represent TRUE and a digital "0" will represent a FALSE. The threshold for the comparator is set at about the mid-point of the total expected excursion of the SSC signal. At some regular and pre-determined rate the digital values in this single bit stream are captured in a array in memory. The rate of capture will not be synchronous with the SSC signal, but will instead be at a regular and steady rate determined by the sampling mechanism in the digital channel of the tester. Logic State Analyzer and Timing Analyzer hardware come to mind as representative of the sort of data capture hardware being described here. Now, if the data capture were synchronous with the SSC signal, then the captured data would lose its time reference and also its SSC (FM) character. However, a steady rate of capture that is at least as fast as the fastest clock rate (which we shall call ω_0) is guaranteed to preserve the underlying FM properties, even though the captured data may from time to time be "missing" a bit. That is, the regular nature of sampling may, at rates lower than twice the highest SSC signal rate, occasionally fail to register a transition. The result is that a string of consecutive 1's might be missing a 0, or a string of consecutive 0's might be missing a 1. The algorithm that we will disclose for recovering the FM modulation profile tolerates this minor wart, and produces the correct answer, anyway. These measured 1's and 0's are held in a first measurement array.

It will further be appreciated that, since the algorithm begins with a threshold operation, the signal whose FM profile is to be measured might be a sinusoidal FM signal having equal plus and minus peak excursions about ground, rather than a digital signal referenced to ground. In such a sinusoidal case, the threshold would be, say, greater than or equal to zero volts. Furthermore, the modulating signal (i.e., the signal whose modulation profile is of interest) need not be sinusoidal; it might, for example, be triangular, or even something else, such a as a saw tooth. Also, and in either case, it is desirable to capture at least three to five cycles of the SSC signal, to allow for settling of certain filtering steps to be described in due course.

To continue, then, if base band and harmonic spectrum information are desired, then the captured data is (at 3) operated upon with a suitable (and conventional) DFT (Discrete Fourier Transform) process, whereupon the results can be analyzed and classified (at 4), after which the spectrum data can be examined and compared to specified limits (5). This right-hand fork of the flow chart 1 may require that the capture rate of the digital channel meet certain Nyquist criteria.

Our main interest is in the left-hand fork of the flow chart 1. Here there are shown a series of steps 6-10 that operate on captured SSC data acquired back at the first step (2). The captured data might be the same data that was used by the right-hand branch of the flow chart, or it might be a data from a separate instance of data acquisition to supply data in the first measurement array.

At this point the reader should refer simultaneously to Figures 1, 2A and 2B. As shown in Figure 2A, the SSC signal under consideration has a certain spectrum 11 (we have not attempted to indicate any harmonics -- just the base band, or fundamental, spectrum itself). It is the FM modulation profile of that spectrum 11 that we wish to measure. The TRANSLATE SPECTRUM step (6) of the flow chart 1 is a software implementation of a heterodyne operation that produces upper and lower side bands (12 and 14, respectively in Figure 2B) that are not only displaced from ω_0, but that are also reflected (or reversed in "direction"). In order to perform this computation we employ the expedient of taking a logical 1 to represent the number one, and taking a logical 0 to represent a the number zero. It will be appreciated that the modulation profile of the sidebands is, save for the reflection, the same as for the original spectrum 11. The TRANSLATE SPECTRUM step (6) operates on the first measurement array and produces in a second translation array a (numerical) string of plus or minus ones and zeros that represents the superposition of the two side bands.

The next step (7) in the algorithm is the application to the data in the second translation array of a FIRST FIR (Finite Impulse Response) FILTER. Its purpose is to act as a low pass filter that removes the upper side band. The output of the FIRST FIR FILTER is a third array that would normally have both original sine and original cosine sections that are populated with fractional values for each sample, save that the values of the cosine terms in the cosine section will be all zeros, and need not actually exist as memory having those zero values. (The absence of any non-zero cosine terms arises from setting the sampling threshold at the mid-point of the excursion of the SSC signal, and at a rate we shall call ω_0 in Figure 2A and 2B. The computations of the FIRST FIR FILTER may be simplified to reflect this fact. If the sampling were accomplished at twice that rate then there would be two interleaved original sequences of sine and non-zero cosine data. That is, if sampling were done at a rate of $2\omega_0$, then the result would be construed as two interleaved sections that were each sampled at ω_0 and that are separated in phase by 90°. What is of note is that the algorithm shown below still works if the data is sampled at just ω_0.)

Following the FIRST FIR FILTER step (7) is a DISCRETE DIFFERENTIATION AND RAW PHASE EXTRACTION STEP (8). The discrete differentiation takes the differences between successive original sine terms in the third array to operationally discover the rate of change of those original sine terms, which is, as is well known, cosine information. If the original cosine terms are not zero, then the same discrete differentiation is also performed for the original cosine terms to find derived sine terms, although this step may be omitted if the original cosine terms are zeros. The original sine terms and their derived cosine terms, and the original (and assumed-to-be-zero) cosine terms and their derived sine terms can all be combined to find phase information, as shown below. The change in the phase ($\Delta\varphi(t)$) can then be used to further find the change in frequency ($\Delta\omega(t)$), also as shown below. It is, of course, the change in frequency that describes the FM modulation profile.

$$S_1(t) = \frac{1}{2}\sin\Delta\varphi(t)$$

$$S_2(t) = \frac{1}{2}\cos\Delta\varphi(t)$$

$$\frac{dS_1(t)}{dt} = \frac{1}{2}\frac{d\sin\Delta\varphi(t)}{dt} = \frac{1}{2}\frac{d\Delta\varphi(t)}{dt}\cos\Delta\varphi(t)$$

$$\frac{dS_2(t)}{dt} = \frac{1}{2}\frac{d\cos\Delta\varphi(t)}{dt} = -\frac{1}{2}\frac{d\Delta\varphi(t)}{dt}\sin\Delta\varphi(t)$$

$$\frac{d\Delta\varphi(t)}{dt} = \Delta\omega(t)$$

$$\cos^2\Delta\varphi(t) + \sin^2\Delta\varphi(t) = 1$$

Therefore

$$\left(\frac{dS_1(t)}{dt}\right)^2 + \left(\frac{dS_2(t)}{dt}\right)^2 = \frac{1}{4}(\Delta\omega(t))^2 \rightarrow \frac{1}{2}\sqrt{(\Delta\omega(t))^2} = \frac{1}{2}|\Delta\omega(t)|$$

The next step is a SECOND FIR FILTER (9). This accepts a fourth array of consecutive **Δω(*t*)** values from step (8) and smooths it to become a fifth array of consecutive values that the FM modulation profile. An example of what such data is like is shown in the graph 15 of Figure 3A. The filtered result is not necessarily perfect, a typical graph 16 of which is shown in Figure 3B; location 17 is a minor imperfection that arises from nature of the binary sampling technique, and might be ameliorated by sampling at a higher rate or by more stringent filtering. Note also that the SECOND FIR FILTER needs some time to settle (i.e., note that the extreme left-hand edge of the graph is misshapen). This is a principle reason why the CAPTURE step (2) ought to acquire three to five cycles of the SSC signal. It will be appreciated that the recovery of different modulation profiles for the SSC signal might benefit from corresponding adjustments to, or changes to the nature of, the SECOND FIR FILTER.

Subsequent to the SECOND FIR FILTER step the sought-after FM modulation characteristics are available and can be examined (10). That is, the fifth array can be treated as a potential graph (e.g., 16 in Figure 3B) that is in tabular form, and inspected for maximum and minimum ordinates (which after suitable scaling represents frequency values) along the abscissa (time).

The following APPENDIX is pseudo code (essentially a complete C++ program) that implements the steps of the left-hand fork of the flow chart of Figure 1. In one embodiment this program executes on the Agilent 93000 SOC System. It is well commented and is believed to be, in light of the foregoing Specification, self-explanatory to those who are skilled in the C++ programming language and also in the basics of digital signal processing techniques, such as the Discrete Fourier Transform. The line numbers that appear to the left of the listing are not part of the program and have appended merely to facilitate references in this Specification to locations in the text of the APPENDIX.

In particular:

Lines 20 - 56 correspond to step (2) of the flow char of Figure 1;
Lines 57 - 72 correspond to step (6) of the flow char of Figure 1;
Lines 73 - 93 correspond to step (7) of the flow char of Figure 1;
Lines 94 - 123 correspond to step (8) of the flow char of Figure 1;
Lines 124 - 137 correspond to step (9) of the flow char of Figure 1;
Lines 138 - 165 correspond to step (10) of the flow char of Figure 1; and
Lines 166 - 220 are a subroutine for the FIR filters of steps (7) and (9).

APPENDIX

```
//------------------------------------------------------------------
//- DESCRIPTION:
//-
//- This pseudo code (which is almost a complete C++ program) shows
//- the extraction of SSC frequency modulation information based on
//- captured single bit digital data which might look like, for
//- example: 101010101010001010101011011010101010101010 .
//-
//- This example routine executes on an Agilent 93000SOC ATE
//- system. It captures data on a single pin under consideration,
//- which will be the SSC output pin or transmitter pin which
//- outputs an SSC signal at high speed. It uses a standard
//- subroutine call available in Agilent93000SOC SW, through which
//- captured data will be retrieved into the workstation.
//- Subsequently, this algorithm will begin to operate on this
//- captured single bit digital data to extract frequency
//- modulation parameters.
//-
//--------------------------------------------------------------

//--------------------------------------------------------------
//- A) Digital capture test:
//-
//- A.1)
//- First an array variable Captured_Data of type  ARRAY_D
//- is defined. ARRAY_D's an one-dimensional array data of
//- DOUBLE type. The system automatically allocates memory
//- by the number of elements assigned to "Captured_Data".
//- The number of array elements is checked and is automatically
//- resized if necessary.
//-
//- A.2)
//- DIGITAL_CAPTURE_TEST() is a standard API ( Application
//- Program Interface) call of 93000 SOC SW to run the test in the
//- digital capture mode, and capture data on the specified
//- digital pin for SSC clocking. The captured data is saved in
//- a memory associated with the pin ("tester pin channel memory').
//-
//- A.3)
//- Retrieve data from the vector memory
//- VECTOR("VectorVariable").getVectors(NumberSamples)  is
//- another standard API call for retrieving data from the
//- tester pin channel memory back into the array variable
//- "Capture_Data" defined before.
//- "VectorVariable"  is data in tester pin channel memory
//- associated with the SSC signal pin, and
//- "NumberSamples" is the number of samples specified.
//-----------------------------------------------------------
  ARRAR_D Capture_Data;
  DIGITAL_CAPTURE_TEST();
  Capture_Data=VECTOR(VectorVariable).getVectors(NumberSamples);

//-----------------------------------------------------------
//- A.4)
//- Put the captured data to graphic window for checking.
//- PUT_DEBUG is an standard API to perform this task.
//-----------------------------------------------------------
  PUT_DEBUG("Pin","RawData",Capture_Data);

//-----------------------------------------------------------
//- B) Spectrum Translation with sin:
//-
//- Only sin data will be calculated, since all cos data is
//- is equal to zero.
//-----------------------------------------------------------
  ARRAY_D AutoWSin,AutoWCos;
  cap_size=Capture_Data.size();
  AutoWSin.resize(cap_size);
  AutoWCos.resize(cap_size);
  INT loopcount,cap_size;
  for (loopcount=0;loopcount<cap_size;loopcount+=2)
  {
      AutoWSin[loopcount]=Capture_Data[loopcount]/2;
      AutoWSin[loopcount+1]=-Capture_Data[loopcount+1]/2;
  }


//-----------------------------------------------------------
//- C) First FIR Filtering:
```

```
//-
//- C.1)
//- First FIR generation:
//- dsp_LPF_FIR is a subroutine for creating coefficients
//- of a finite impulse response filter.
//-
//- C.2)
//- Applying this filter to data:
//- DSP_CONVOL is a standard API call to perform convolution
//- operation of two arrays and result is saved in FIRWSin
//- and FIRWCos. FIRWCos in fact are all zero.
//-------------------------------------------------------------
  INT Nfir=1024;
  DOUBLE dLPBW=0.05;
  ARRAY_D dFIR;
  dsp_LPF_FIR(Nfir,dLPBW,dFIR);

  ARRAY_D FIRWSin,FIRWCos;
  DSP_CONVOL(dFIR,AutoWSin,FIRWSin);
  DSP_CONVOL(dFIR,AutoWCos,FIRWCos);

//-------------------------------------------------------------
//- D) Discrete Differentiation and Raw Phase Extraction
//-
//- D.1) Discrete Differentiation based on FIRWSin(i)-FIRWS(i-1)
//-      and FIRWCos(i)-FIRWCos(i-1) where FIRWSin and FIRWCos
//-      are the sinusoidal and cosinusoidal components of the
//-      data array from the first FIR filter result.
//-
//- D.2) Squaring of each differentiation result
//-
//- D.3) Summation of these two squaring results and root
//-      squaring afterwards
//-------------------------------------------------------------
  ARRAY_D SSC_Value(Capture_Data.size());
  DOUBLE Tmp_Value;
  for (count=0;count<points;count++)
  {
      if (count==0)
      {
         SSC_Value[count]=0.0;
      }
      else
      {
Tmp_Value=(FIRWSin[count]-FIRWSin[count-1])*(FIRWSin[count]-FIRWSin
[count-1]);
Tmp_Value+=(FIRWCos[count]-FIRWCos[count-1])*(FIRWCos[count]-FIRWCo
s[count-1]);
         SSC_Value[count]=sqrt(Tmp_Value)/Step/2/M_PI*2;
      }
  }

//---------------------------------------------------------------
//- E) Second FIR Filter:
//-
//- E.1)
//-Second FIR Generation:
//-
//- E.2)
//- Filtering with the second FIR filter.
//- After the filtering the desired FM parameters are present.
//---------------------------------------------------------------
  ARRAY_D SSC_Demod;
  dLPBW=0.000005;
  dsp_LPF_FIR(Nfir,dLPBW,dFIR);
  DSP_CONVOL(dFIR,SSC_Value,SSC_Demod);

//---------------------------------------------------------------
//- F) Analysis of the result
//-
//- Find the frequency data from the changing phase data. Find
//- the minimum and maximum values of the frequency modulation.
//---------------------------------------------------------------

//---------------------------------------------------------------
//- DSP_MINMAX is a standard API call of 93000 SOC SW
//- to find the min and max value in an array. The variables
//- min and max are proportional to the min and max frequency.
//---------------------------------------------------------------
  DOUBLE min,max;
  DSP_MINMAX(SSC_Demod,&min,&max);
```

```
//-----------------------------------------------------------------
//- DSP_EDGE is a standard API call of 93000 SOC SW
//- to find the element location of rise/fall in an array
//- with specified threshold.
//- First find edge1 at 50% threshold and then find the next
//- edge2 at 50% threshold. The difference is then between
//- edge2 and edge1 and result is saved in period.
//-----------------------------------------------------------------
  DOUBLE edge1,edge2;
  DOUBLE period;
  int start;
  start=1000;
  DSP_EDGE(SSC_Demod, &edge1, start, 0.5*(max-min)+min; RISE);
  DSP_EDGE(SSC_Demod, &edge2, edge1, 0.5*(max-min)+min; RISE);
  period=edge2-edge1;

//-----------------------------------------------------------------
// End of the main routine.
//-----------------------------------------------------------------

//-----------------------------------------------------------------
// Subroutine for finite impulse response filter coefficients:
//-----------------------------------------------------------------
void dsp_LPF_FIR(
   INT        Nfir,       /* Length of FIR                          */
   DOUBLE     dBW,        /* Pass band width of LPF (1.0 = Fdgt)    */
   ARRAY_D & dFIR         /* LPF FIR array data                     */
)
{
  INT             i,Nsp,Nc,Ncycles;
  ARRAY_COMPLEX CSp,CWave;
  ARRAY_D         dWave1,dWave2;
  ARRAY_D         dSp1,dSp2;
  DOUBLE          dX,dNc,dP;
  dNc=Nfir*dBW;             /* Passband Ratio (relative to dFdgt)    /
  Nc=(int)(dNc+0.5);
  Nsp=Nfir/2;
  CSp.resize(Nfir);
  for (i=0;i<=Nc;i++)
  {
      CSp[i].real()=1.0;
```

```
        CSp[i].imag()=0.0;
    }
    for (i=Nc+1;i<=Nsp;i++)
    {
        CSp[i].real()=0.0;
        CSp[i].imag()=0.0;
    }
    for (i=1;i<Nsp;i++)
    {
        CSp[Nfir-i].real()= CSp[i].real();
        CSp[Nfir-i].imag()=-CSp[i].imag();
    }
    DSP_IFFT(CSp,CWave);
    dFIR.resize(Nfir);
    for (i=0;i<Nfir;i++) dFIR[i]=0.5*CWave[i].real();
    for (i=0;i<Nsp;i++)
    {
        dX=dFIR[i];
        dFIR[i]=dFIR[Nsp+i];
        dFIR[Nsp+i]=dX;
    }
    /* Hanning weighting onto the FIR */
    for (i=0;i<Nfir;i++)
    {
        dX=0.5*(1.0-cos(2.0*M_PI*(double)i/(double)(Nfir-1)));
        dFIR[i]=dFIR[i]*dX;
    }
    /* Scaling */
    dX=0.0; for (i=0;i<Nfir;i++) dX=dX+dFIR[i];
    DSP_MUL_SCL(1.0/dX,dFIR,dFIR);
}
```